Today

- Answer more questions on:
  - Midterm
  - Homework for Unit B
- Talk about variable, expressions and functions in the context of Netlogo and Javascript.
- Remind you to do the readings for Units D, E and F.

Netlogo as Computer Science

- Here’s a small piece of Netlogo:
  ```
  to catch-sheep
    let prey one-of (sheep-here with [not grabbed?])
    if prey != nobody
      [ set grabbed?-of prey true
        ask prey [ die ]
        set energy energy + wolf-gain-from-food
      ]
  end
  ```

Any questions?
What's going on?

- This is something that we tell a wolf to do in the wolf/sheep model.
- We tell it to make “prey” one of the sheep on the patch that the wolf is on, one which is not “grabbed?”.
- If there is such a sheep, we tell the wolf:
  - mark it as “grabbed?”
  - make the sheep die
  - get more energy from eating the sheep.
- What does the computer have to do to make this work?

Variables

- One thing we need is for each sheep to know whether it has been “grabbed?” or not.
- Why is this important?
- So that the sheep can tell, we give it a local variable:
  ```
  sheep-own [ grabbed? ]
  ```
  it is “local” because each sheep has its own — it is local to the individual sheep.
- Since “grabbed?” belongs to the sheep, when the wolf changes it, it has to use:
  ```
  set grabbed?-of prey true
  ```
  rather than:
  ```
  set grabbed? true
  ```

Variables (more)

- We say that the scope of the variable grabbed? is the sheep.
- Because the variable is local, not every agent can access it.
- Indeed, only a single sheep can access each grabbed?.
- In contrast, this is a global variable:
  ```
  globals [ ticks ]
  ```
- Since ticks is global, any agent can find out the value of ticks.
Variables (even more)

- Here are some more local variables:
  - turtles-own [ energy ]
  - patches-own [ countdown ]

- Other variables that you have come across are pcolor, xcor, ycor, pxcor, pycor.

- In general, variables give us a way to store values.

Variables (last)

- Our example from wolf/sheep shows another kind of local variable.
  - let prey one-of (sheep-here with [not grabbed?])

- Here prey is a variable that is local to catch-sheep, rather than to any turtle.

- The scope of prey is catch-sheep.

- Trying to access prey from outside catch-sheep will give an error.

Expressions (more)

- Wolf/sheep also gives us these examples
  - set pcolor green
  - set color black
  - rt random-float 50 - random-float 50
  - set energy random (2 * wolf-gain-from-food)

- What is going on in these?
Expressions (even more)

- `set` is an example of assignment. It changes the value of a variable.
  - `set energy energy + wolf-gain-from-food`
    changes the value of the variable `energy` to be the old value of `energy` plus the value of `wolf-gain-from-food`.
- `set color black`
  changes the value of the variable `color` to be `black`.

Expressions (last)

- `rt random-float 50 - random-float 50`
  - `set energy random (2 * wolf-gain-from-food)`
    make use of functions, random and random-float.

Functions (first time)

- Functions are bits of program that generate values.
- Since they generate values, it is natural that we use them along with assignment.
- We use functions as a way to get abstraction.
- You can think of abstraction as “hiding the detail”.
- Rather than writing out the Netlogo code for generating a random number every time that we want one, we just call random.
- `random` is provided by the folk who wrote Netlogo, but you can also write your own functions.

Procedures

- In fact we don’t write many of our own functions in Netlogo.
- We do write procedures. Procedures are bits of code that do something:
  - `catch-sheep` is a nice example.
    to catch-sheep
      let prey one-of (sheep-here with [not grabbed?])
      if prey != nobody
        [ set grabbed?-of prey true
          ask prey [ die ]
          set energy energy + wolf-gain-from-food
        ]
      end
Procedures (more)

- A procedure starts with:
  ```
  to name-of-procedure
  ```

- and ends with
  ```
  end
  ```

- In between, the procedure contains a list of instructions.
- These instructions are the steps in the algorithm that the procedure uses.

Procedures (even more)

- You then call a procedure to make it execute.
  ```
  to go:
    ask wolves [ movesetenergy energy - 1
    catch-sheep
    reproduce-wolves
    death ]
  end
  ```

- So one procedure is called by a second procedure which may be called by a third procedure, and so on . . . .

Procedures (last)

- Procedures can take inputs:
  ```
  to draw-polygon [num-sides len]
    pd
    repeat num-sides
      [ fd len
        rt (360 / num-sides) ]
  end
  ```

- To call this procedure, you have to give it a number (an integer) that sets the number of sides, and another number (integer) that sets the length of the sides.

  ```
  ask turtles [ draw-polygon 8 who ]
  ```

Functions (again)

- In Netlogo, functions are called reporters.
- They report values.
- They are defined and called much like procedures:
  ```
  to-report absolute-value [number]
    ifelse number >= 0
      [ report number ]
      [ report 0 - number ]
  end
  ```
Summary

- This lecture talked about some of the computer science ideas behind Netlogo.
  - Variables
  - Expressions
  - Functions